

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 09-069082

(43)Date of publication of application : 11.03.1997

(51)Int.Cl. G06F 15/16

(21)Application number : 07-341839

(71)Applicant : TOSHIBA CORP

(22)Date of filing : 27.12.1995

(72)Inventor : HOSHINA SATOSHI
SAKAI HIROSHI
HIRAYAMA HIDEAKI
OMORI TAKASHI
MASUBUCHI YOSHIO
FUJII TAKAHIRO

(30)Priority

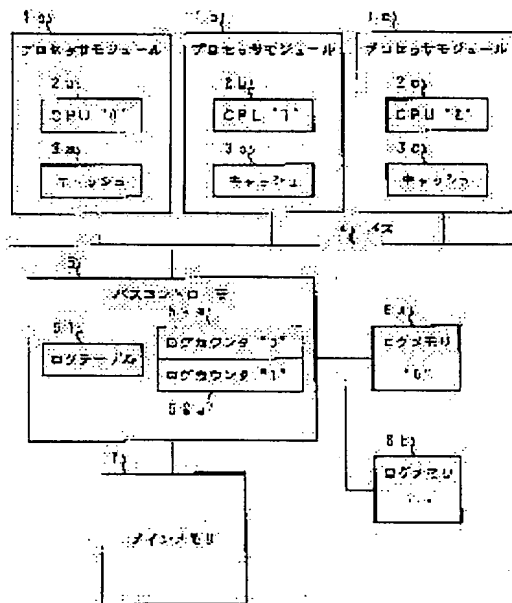
Priority number : 07151735 Priority date : 19.06.1995 Priority country : JP

(54) MULTIPROCESSOR SYSTEM

(57)Abstract:

PROBLEM TO BE SOLVED: To provide the computer system which can greatly decrease stand-by states in the whole system at check point acquisition.

SOLUTION: This system is equipped with a plurality of log memories (log memories 6a-6b) which hold the update history of a main memory 7 and CPUs 2a-2c record the update history of the main memory 7 in one of the log memories 6a-6b; when the internal state and the contents of a cache memory are written out to the main memory 7 in response to check point acquisition, switching is so performed that CPUs having finished writing them out record the update history of the main memory 7 in another log memory, thereby restarting ordinary data processing without waiting for other CPUs to have acquired check points.

**LEGAL STATUS**

[Date of request for examination] 06.03.1997

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number] 3085899

[Date of registration] 07.07.2000

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

BEST AVAILABLE COPY

(19)日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開平9-69082

(43)公開日 平成9年(1997)3月11日

(51)Int.Cl.⁶

G 0 6 F 15/16

識別記号

4 3 0

庁内整理番号

F I

G 0 6 F 15/16

技術表示箇所

4 3 0 B

審査請求 未請求 請求項の数17 O L (全 21 頁)

(21)出願番号 特願平7-341839

(22)出願日 平成7年(1995)12月27日

(31)優先権主張番号 特願平7-151735

(32)優先日 平7(1995)6月19日

(33)優先権主張国 日本 (J P)

(71)出願人 000003078

株式会社東芝

神奈川県川崎市幸区堀川町72番地

(72)発明者 保科 聡

東京都青梅市末広町2丁目9番地 株式会
社東芝青梅工場内

(72)発明者 酒井 浩

東京都青梅市末広町2丁目9番地 株式会
社東芝青梅工場内

(72)発明者 平山 秀昭

東京都青梅市末広町2丁目9番地 株式会
社東芝青梅工場内

(74)代理人 弁理士 鈴江 武彦

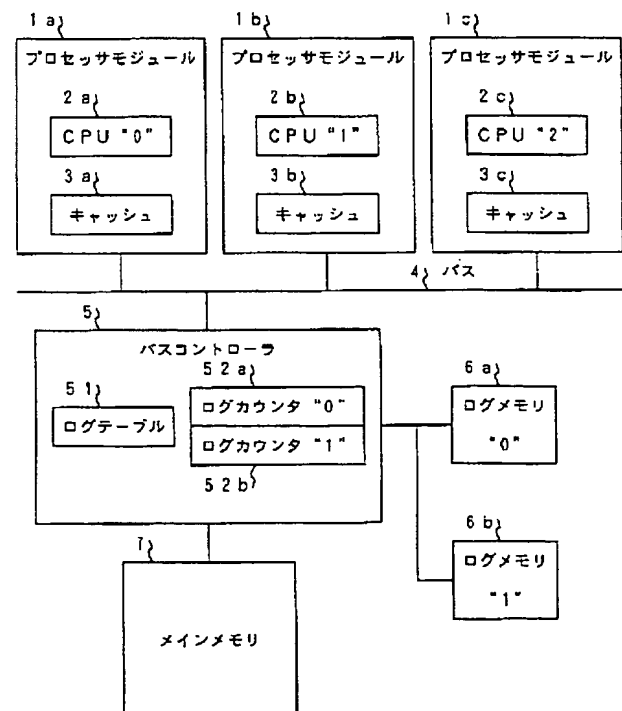
最終頁に続く

(54)【発明の名称】 マルチプロセッサシステム

(57)【要約】

【目的】チェックポイント取得時におけるシステム全体の待機状態を大幅に減少させることを可能とする計算機システムを提供する。

【構成】メインメモリ7の更新履歴を保持するログメモリを複数備え(ログメモリ6a~6b)、CPU2a~2cがその複数のログメモリ6a~6bの中のいずれか一つにメインメモリ7の更新履歴を記録していき、チェックポイント取得に伴ってその内部状態およびキャッシュメモリの内容をメインメモリ7に書き出す際に、その書き出しを完了したCPUから他のいずれかのログメモリにメインメモリの更新履歴を記録するように切り替えて、他のCPUのチェックポイント取得完了を待機せずに通常のデータ処理を再開することを特徴とする。



【特許請求の範囲】

【請求項1】 キャッシュメモリを各々が有する複数のプロセッサと、これらプロセッサによって共有される共有メモリと、この共有メモリの更新履歴を格納するログメモリとを有するマルチプロセッサシステムであって、前記複数のプロセッサそれぞれがチェックポイント毎にその内部状態、および前記共有メモリに反映されてない前記キャッシュメモリの内容を前記共有メモリに書き込み、前記マルチプロセッサシステムの障害発生時に、前記ログメモリに格納された更新履歴を使用することによって前記共有メモリを前記障害発生前のチェックポイントの時点に復元し、その時点から処理を再開するマルチプロセッサシステムにおいて、前記共有メモリの更新履歴を示す履歴情報を格納する第1および第2のログメモリと、これら第1および第2のログメモリの中で履歴情報の格納に使用するログメモリを前記プロセッサ毎に選択するログメモリ選択手段とを具備し、前記各プロセッサは、その内部状態およびキャッシュメモリのデータを前記共有メモリに書き込むチェックポイント取得処理を完了したとき、前記ログメモリ選択手段を制御することによって、そのプロセッサが使用するログメモリをそれまで使用していた前記第1および第2のログメモリの一方から他方に切り替え、前記チェックポイント取得処理を実行するために中断したプロセスの実行を再開することを特徴とするマルチプロセッサシステム。

【請求項2】 前記複数のプロセッサの中の少なくとも1つのプロセッサのキャッシュメモリのライトポリシーをライトバックからライトスルーに変更し、そのライトスルーキャッシュを持つプロセッサにリアルタイムプロセスを実行させる手段をさらに具備し、前記ライトスルーキャッシュを持つプロセッサは、前記チェックポイントにおいてその内部状態だけを前記共有メモリに書き込むチェックポイント取得処理を実行し、それが完了したとき、前記ログメモリ選択手段を制御することによって、そのプロセッサが使用するログメモリをそれまで使用していた前記第1および第2のログメモリの一方から他方に切り替え、前記チェックポイント取得処理を実行するために中断したリアルタイムプロセスの実行を再開することを特徴とするマルチプロセッサシステム。

【請求項3】 前記第1および第2のログメモリは、1つの物理メモリと、前記第1のログメモリにおける更新履歴格納番地を指定するポインタ値を保持する第1のカウントであって、前記物理メモリの先頭番地から最終番地に向かって前記ポインタ値が順次インクリメントされる第1のカウントと、前記第2のログメモリにおける更新履歴格納番地を指定

2

するポインタ値を保持する第2のカウントであって、前記物理メモリの最終番地から先頭番地に向かって前記ポインタ値が順次デクリメントされる第2のカウントとを具備することを特徴とする請求項1記載のマルチプロセッサシステム。

【請求項4】 前記ログメモリ選択手段は、前記複数のプロセッサそれぞれからのログメモリ切り替え要求に応じて書き換え可能に構成され、前複数のプロセッサとそれらプロセッサが使用するログメモリとの対応関係を示す情報が格納されるログテーブルと、前記共有メモリにデータ書き込みを行ったプロセッサが使用するログメモリを前記ログテーブルを参照して決定し、そのログメモリに対応する前記第1および第2のカウントの一方のポインタ値を選択する手段とを具備することを特徴とする請求項3記載のマルチプロセッサシステム。

【請求項5】 前記第1および第2のログメモリは、前記履歴情報を格納する第1フィールドと前記共有メモリにデータ書き込みを行ったプロセッサが現在使用しているログメモリを示すログメモリ識別子を格納する第2フィールドとを各々が含む複数のエントリを有する1つの物理メモリを具備し、前記各プロセッサによるログメモリの切り替えにตอบสนองして、前記第2フィールドに格納されるログメモリ識別子の値が変更されることを特徴とする請求項1記載のマルチプロセッサシステム。

【請求項6】 前記ログメモリ選択手段は、前記複数のプロセッサそれぞれからのログメモリ切り替え要求に応じて書き換え可能に構成され、前複数のプロセッサとそれらプロセッサが使用するログメモリとの対応関係を示す情報が格納されるログテーブルと、前記共有メモリにデータ書き込みを行ったプロセッサが使用するログメモリを前記ログテーブルを参照して決定し、そのログメモリに対応するログメモリ識別子を、前記書き込みに対応する履歴情報を格納すべき前記物理メモリのエントリの第2フィールドに格納する手段とを具備することを特徴とする請求項5記載のマルチプロセッサシステム。

【請求項7】 前記第1および第2の双方のログメモリにおける履歴情報格納番地を指定するポインタ値を保持する第1のカウントであって、前記物理メモリの先頭エントリから最終エントリに向かって前記ポインタ値が順次インクリメントされる第1のカウントと、前記複数のプロセッサの中で最初にログメモリを切り替えたプロセッサが使用を開始した前記物理メモリの最初のエントリを示すポインタ値を保持する第2のカウントとをさらに具備し、前記マルチプロセッサの障害発生時は、前記第2のカウントのポインタ値が示すエントリから前記第1のカウントのポインタ値が示すエントリの中で、前記第1のカウ

ンタのポインタ値が示すエントリの中に格納されているログメモリ識別子と同一のログメモリ識別子が格納されているエントリの履歴情報が前記共有メモリの復元のために使用されることを特徴とする請求項5記載のマルチプロセッサシステム。

【請求項8】 キャッシュメモリを各々が有する複数のプロセッサと、これらプロセッサによって共有される共有メモリと、この共有メモリの更新履歴を格納するログメモリとを有するマルチプロセッサシステムであって、前記複数のプロセッサそれぞれがチェックポイント毎にその内部状態、および前記共有メモリに反映されてない前記キャッシュメモリの内容を前記共有メモリに書き込み、前記マルチプロセッサシステムの障害発生時に、前記ログメモリに格納された更新履歴を使用することによって前記共有メモリを前記障害発生前のチェックポイントの時点に復元し、その時点から処理を再開するマルチプロセッサシステムにおいて、

前記共有メモリの更新履歴を格納する第1および第2のログメモリと、

これら第1および第2のログメモリの中で更新履歴の保存に使用するログメモリを前記プロセッサ毎に選択するログメモリ選択手段と、

前記複数のプロセッサそれぞれのキャッシュメモリのライトポリシーをライトバックまたはライトスルーに設定する手段であって、前記マルチプロセッサシステムの動作状態を監視し、そのシステムのスループットとレスポンスタイムとに基づいて、ライトスルーキャッシュとして動作するキャッシュメモリを持つプロセッサ数を動的に変更する手段とを具備し、

前記ライトバックキャッシュとして動作するキャッシュメモリを持つプロセッサは、その内部状態およびキャッシュメモリのデータを前記共有メモリに書き込むチェックポイント取得処理を実行し、それが完了したとき、前記ログメモリ選択手段を制御することによって、そのプロセッサが使用するログメモリをそれまで使用していた前記第1および第2のログメモリの一方から他方に切り替え、前記チェックポイント取得処理を実行するために中断したプロセスの実行を再開し、

前記ライトスルーキャッシュとして動作するキャッシュメモリを持つプロセッサは、その内部状態だけを前記共有メモリに書き込むチェックポイント取得処理を実行し、それが完了したとき、前記ログメモリ選択手段を制御することによって、そのプロセッサが使用するログメモリをそれまで使用していた前記第1および第2のログメモリの一方から他方に切り替え、前記チェックポイント取得処理を実行するために中断したプロセスの実行を再開することを特徴とするマルチプロセッサシステム。

【請求項9】 前記第1および第2のログメモリは、1つの物理メモリと、

前記第1のログメモリにおける更新履歴格納番地を指定

するポインタ値を保持する第1のカウンタであって、前記物理メモリの先頭番地から最終番地に向かって前記ポインタ値が順次インクリメントされる第1のカウンタと、

前記第2のログメモリにおける更新履歴格納番地を指定するポインタ値を保持する第2のカウンタであって、前記物理メモリの最終番地から先頭番地に向かって前記ポインタ値が順次デクリメントされる第2のカウンタとを具備することを特徴とする請求項8記載のマルチプロセッサシステム。

【請求項10】 前記ログメモリ選択手段は、前記複数のプロセッサそれぞれからのログメモリ切り替え要求に応じて書き換え可能に構成され、前複数のプロセッサとそれらプロセッサが使用するログメモリとの対応関係を示す情報が格納されるログテーブルと、前記共有メモリにデータ書き込みを行ったプロセッサが使用するログメモリを前記ログテーブルを参照して決定し、そのログメモリに対応する前記第1および第2のカウンタの一方のポインタ値を選択する手段とを具備することを特徴とする請求項9記載のマルチプロセッサシステム。

【請求項11】 前記第1および第2のログメモリは、前記履歴情報を格納する第1フィールドと前記共有メモリにデータ書き込みを行ったプロセッサが現在使用しているログメモリを示すログメモリ識別子を格納する第2フィールドとを各々が含む複数のエントリを有する1つの物理メモリを具備し、

前記各プロセッサによるログメモリの切り替えにตอบสนองして、前記第2フィールドに格納されるログメモリ識別子の値が変更されることを特徴とする請求項8記載のマルチプロセッサシステム。

【請求項12】 前記ログメモリ選択手段は、前記複数のプロセッサそれぞれからのログメモリ切り替え要求に応じて書き換え可能に構成され、前複数のプロセッサとそれらプロセッサが使用するログメモリとの対応関係を示す情報が格納されるログテーブルと、前記共有メモリにデータ書き込みを行ったプロセッサが使用するログメモリを前記ログテーブルを参照して決定し、そのログメモリに対応するログメモリ識別子を、前記書き込みに対応する履歴情報を格納すべき前記物理メモリのエントリの第2フィールドに格納する手段とを具備することを特徴とする請求項11記載のマルチプロセッサシステム。

【請求項13】 前記第1および第2の双方のログメモリにおける履歴情報格納番地を指定するポインタ値を保持する第1のカウンタであって、前記物理メモリの先頭エントリから最終エントリに向かって前記ポインタ値が順次インクリメントされる第1のカウンタと、前記複数のプロセッサの中で最初にログメモリを切り替えたプロセッサが使用を開始した前記物理メモリの最初

のエントリを示すポインタ値を保持する第2のカウンタとをさらに具備し、

前記マルチプロセッサの障害発生時は、前記第2のカウンタのポインタ値が示すエントリから前記第1のカウンタのポインタ値が示すエントリの中で、前記第1のカウンタのポインタ値が示すエントリの中に格納されているログメモリ識別子と同一のログメモリ識別子が格納されているエントリの履歴情報が前記共有メモリの復元のために使用されることを特徴とする請求項1記載のマルチプロセッサシステム。

【請求項14】 それぞれにキャッシュメモリを備えたCPUを複数有し、それらのCPUがメインメモリを共有するマルチプロセッサシステムであって、上記メインメモリの更新時にそのアドレスと更新前の内容とを更新履歴として保持するログメモリを備えるとともに、上記各CPUが同期して所定のタイミングで周期的にその内部状態および必要に応じてキャッシュメモリの内容を上記メインメモリに書き出すことにより、上記マルチプロセッサシステムに障害が発生したときに、上記ログメモリに保持された更新履歴により上記メインメモリを所定の時点に復元して、その障害発生前の状態から、再実行することを實現するマルチプロセッサシステムにおいて、

上記ログメモリを複数備え、
上記各CPUがその複数のログメモリの中のいずれか一つに上記メインメモリの更新履歴を記録していき、上記特定のタイミングでその内部状態および必要に応じてキャッシュメモリの内容を上記メインメモリに書き出す際に、その書き出しを完了したCPUは、他のいずれかのログメモリにメインメモリの更新履歴を記録するように切り替えて、即座に通常のデータ処理を再開することを特徴とするマルチプロセッサシステム。

【請求項15】 CPUから書き込み要求があったときに、即座にメインメモリに書き込むライトスルー方式のキャッシュメモリを備えたCPUと、一旦キャッシュメモリ内に蓄えて適宜メインメモリに書き込むライトバック方式のキャッシュメモリを備えたCPUとを上記マルチプロセッサシステムに混在させ、常に所定の時間内に終了させなければならない処理を上記ライトスルー方式のキャッシュメモリを備えたCPUにて実行することを特徴とする請求項14記載のマルチプロセッサシステム。

【請求項16】 上記各CPUの備えるキャッシュメモリは、ライトスルー方式およびライトバック方式のいずれの方式で動作するのか上記マルチプロセッサシステム上で稼働するプログラムの指令により動的に切り替えることを特徴とする請求項15記載のマルチプロセッサシステム。

【請求項17】 上記マルチプロセッサシステム上で稼働するプログラムは、システムの稼働状態を監視して、

実行されるプログラムのスループットおよび応答性を測定し、ライトスルー方式およびライトバック方式それぞれ適切な数を決定する手段を含むことを特徴とする請求項16記載のマルチプロセッサシステム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は共有メモリを有するマルチプロセッサシステムに関し、特にログメモリを使用したチェックポイントリスタート方法を採用したマルチプロセッサシステムに関する。

【0002】

【従来の技術】一般に、フォールトトレラントコンピュータシステムの構成法のひとつとして、チェックポイントリスタート方法を用いたシステム回復技術が知られている。このチェックポイントリスタート方法では、プロセスの再実行に必要な情報が定期的に保存される。この情報を保存する時点をチェックポイントと呼び、その情報を保存することをチェックポイントの取得と呼ぶ。コンピュータシステムの障害発生によってプログラム実行が中断されたとき、その中断されたプロセスの状態は障害発生前のチェックポイントまでロールバックされ、そのチェックポイントから再実行される。

【0003】チェックポイント取得処理で保存される情報には、プロセッサの内部状態、キャッシュメモリ、メインメモリ等がある。最近では、チェックポイント取得処理に要する時間、つまり通常のプログラム実行が中断される時間を短くするために、ログメモリを使用したチェックポイントリスタート方法が採用され始めている。

【0004】この方法では、あるチェックポイントから次のチェックポイントまでの期間において、プロセス実行に伴ってメインメモリが更新される度にその更新前データなどが更新履歴情報としてログメモリに採取される。コンピュータシステムに障害が発生したとき、ログメモリの内容によってメインメモリは障害発生前のチェックポイントの時点に復元される。従って、ログメモリを使用したチェックポイントリスタート方法を採用した場合には、チェックポイント取得処理で保存しなければならない情報はプロセッサの内部状態とキャッシュメモリだけとなり、チェックポイント取得処理に要する時間を比較的短くすることができる。このようなログメモリを使用したチェックポイントリスタート方法を採用したコンピュータシステムの典型的な構成例を図18に示す。

【0005】このシステムは、プロセッサモジュール100、バス400、バスコントローラ500、ログメモリ600、およびメインメモリ700を備えている。プロセッサモジュール100は、CPUとキャッシュメモリを備えている。

【0006】CPUがメインメモリ700上のデータを更新する時、バスコントローラ500は、その更新履歴

をログメモリ600に記録する。このログメモリ600の残記憶容量がある一定サイズを下回ったとき、CPUは、チェックポイント取得処理を開始し、その内部状態とキャッシュメモリの内容をメインメモリ700に保存する。このチェックポイント取得処理が完了した時点でログメモリ600に保存された更新履歴情報は不要となる。そして、ログメモリ600が初期化されてその更新履歴情報が破棄された後、CPUによって再び通常のプログラム実行処理が再開される。通常のプログラム実行処理が再開されてから次のチェックポイントまでの期間におけるメインメモリ700の更新履歴は、ログメモリ600に保存される。

【0007】

【発明が解決しようとする課題】しかしながら、このようなログメモリを使用したチェックポイントリスタート方法を、メインメモリ700を共有する複数のプロセッサモジュールを含むマルチプロセッサシステムに採用すると、各プロセッサは、自身のチェックポイント取得処理を完了しても、他の全てのプロセッサがチェックポイント取得処理を完了するまで、通常のプログラム実行処理を再開できないという問題が生じる。以下、マルチプロセッサシステムにおけるチェックポイント取得処理に関する問題点を図19を参照して詳述する。

【0008】ここでは、3つのCPU、CPU“0”、CPU“1”、CPU“2”がメインメモリ700を共有するマルチプロセッサシステムを例にとって説明する。

【0009】いま、これらCPU“0”、CPU“1”、CPU“2”は、それぞれ通常のプログラム実行処理を行なっているものとする（図19の（1））。その後、ログメモリの残容量が予め設定された所定量を下回ったとき（図19の（2））、CPU“0”、CPU“1”、CPU“2”それぞれは、その旨を検知してチェックポイント取得処理を開始する（図19の（3））。

【0010】このチェックポイント取得処理の開始タイミングおよび完了までに要する時間は、その検知タイミングや、検知したときに処理中のプログラムの種類、キャッシュメモリの状態などにより異なる。このため、チェックポイント取得処理を完了するタイミングは、CPU毎に異なる。図19では、CPU“0”、CPU“2”、CPU“1”の順でチェックポイント取得処理が完了される例が示されている。

【0011】この場合、CPU“1”のチェックポイント取得処理が完了するまでは、CPU“0”、CPU“2”は通常のプログラム処理を再開することができず、ウェイト状態となる（図19の（4））。なぜなら、全てのCPUのチェックポイント取得処理が完了する時点（図19の（5））までは、障害発生に備えて、ログメモリ600に記録された更新履歴をそのまま保持

しておく必要があるためである。もし全てのCPUのチェックポイント取得処理が完了する前にあるCPUが通常のプログラム実行処理を再開してしまったならば、そのプログラム実行処理に伴うメモリ更新によって、ログメモリ600の内容が書き換えられてしまう。この場合、もし障害が発生すると、その障害発生前のチェックポイントの時点にメインメモリを正常に復元することはできなくなる。

【0012】従って、CPU“0”、CPU“1”、CPU“2”は、それら全てのCPUのチェックポイント取得処理が完了した後に、通常のプログラム実行処理を再開する（図19の（6））。

【0013】このように、マルチプロセッサシステムにおいては、各プロセッサは、自身のチェックポイント取得処理を完了しても、他の全てのプロセッサがチェックポイント取得処理を完了するまで、通常のプログラム実行処理を再開できないという問題がある。また、最近では、キャッシュメモリの大容量化等により、各プロセッサのチェックポイント取得処理そのものに要する時間も長くなる傾向にある。本発明は上記実情に鑑みてなされたものであり、各プロセッサが自身のチェックポイント取得処理を完了した時点で即座に通常のプログラム実行処理を再開できるようにし、システム全体の待機時間を大幅に減少させることができるマルチプロセッサシステムを提供することを目的とする。

【0014】

【課題を解決するための手段】本発明は、キャッシュメモリを各々が有する複数のプロセッサと、これらプロセッサによって共有される共有メモリと、この共有メモリの更新履歴を格納するログメモリとを有するマルチプロセッサシステムであって、前記複数のプロセッサそれぞれがチェックポイント毎にその内部状態、および前記共有メモリに反映されてない前記キャッシュメモリの内容を前記共有メモリに書き込み、前記マルチプロセッサシステムの障害発生時に、前記ログメモリに格納された更新履歴を使用することによって前記共有メモリを前記障害発生前のチェックポイントの時点に復元し、その時点から処理を再開するマルチプロセッサシステムにおいて、前記共有メモリの更新履歴を示す履歴情報を格納する第1および第2のログメモリと、これら第1および第2のログメモリの中で履歴情報の格納に使用するログメモリを前記プロセッサ毎に選択するログメモリ選択手段とを具備し、前記各プロセッサは、その内部状態およびキャッシュメモリのデータを前記共有メモリに書き込むチェックポイント取得処理を完了したとき、前記ログメモリ選択手段を制御することによって、そのプロセッサが使用するログメモリをそれまで使用していた前記第1および第2のログメモリの一方から他方に切り替え、前記チェックポイント取得処理を実行するために中断したプロセスの実行を再開することを特徴とする。

【0015】このマルチプロセッサシステムにおいては、2つのログメモリが設けられており、プロセッサ毎にそれらログメモリを切り替えて使用することができる。このため、チェックポイント取得処理を完了したプロセッサの順で、使用するログメモリをそれまで使用していたログメモリから他方のログメモリに切り替えることにより、それまで使用していたログメモリの内容を破壊することなく、通常のプログラム実行処理を再開することができる。よって、各プロセッサが自身のチェックポイント取得処理を完了した時点で即座に通常のプログラム実行処理を再開できるようになり、システム全体の待機時間を大幅に減少させることができる。2つのログメモリは、1つの物理メモリ上に論理的に実現することが好ましい。これにより、ハードウェアの追加を招くことなく、低コストで高性能のシステムを実現できる。

【0016】また、一部のプロセッサに備えられるキャッシュメモリをライトスルー方式にすることにより、そのプロセッサのチェックポイント取得時間を大幅に削減でき、たとえば、個々の処理についてのレスポンスは多少悪化させることとなっても、常に所定の時間内に終了させる必要がある処理（リアルタイム処理）などをこのプロセッサで処理させるようにすれば、チェックポイント取得に伴う弊害を回避することが可能となる。

【0017】なお、ここでいうライトスルー方式のキャッシュメモリとは、プロセッサから書き込み要求があったときに、即座にメインメモリに書き込みを行なうタイプのキャッシュメモリをいい、また、プロセッサから書き込み要求があったときに、一旦キャッシュメモリ内に蓄えて、必要に応じてメインメモリに書き込みを行なうタイプのキャッシュメモリをライトバック方式のキャッシュメモリという。

【0018】チェックポイント取得に費やす時間を分析調査すると、キャッシュメモリ内の更新データのメインメモリへの書き戻し、すなわちキャッシュフラッシュに費やす時間が大部分を占めることがわかる。したがって、このライトスルー方式のキャッシュメモリを備えたプロセッサのチェックポイント処理は、ライトバック方式のキャッシュメモリを備えたプロセッサのチェックポイント処理と比較して、非常に短時間で終了することがわかる。

【0019】すなわち、このライトスルー方式のキャッシュメモリとライトバック方式のキャッシュメモリとを適切な数でシステム内に混在させることにより、バストラフィックを増やすことなく、且つ全体のパフォーマンスを低下させずに、リアルタイム処理を実行することができる。

【0020】

【発明の実施の形態】

（第1実施形態）図1はこの発明の第1の実施形態に係わるマルチプロセッサシステムの構成を示す図である。

【0021】図1に示されているように、このマルチプロセッサシステムは、メインメモリ7を共有する3つのプロセッサモジュール1a～1cを備えている。これらプロセッサモジュール1a～1cはバス4に接続されており、それぞれCPU2a～2cと、キャッシュメモリ3a～3cとから構成されている。これらキャッシュメモリ3a～3cは、それぞれCPU2a～2cの一次キャッシュ、または二次キャッシュとして動作する。各プロセッサモジュールでは、CPUがメインメモリ7からデータの読み出しを要求した際に、まずキャッシュメモリの中に該当するデータが保持されているかどうかを検査する。ここでもし、キャッシュメモリ内に該当するデータが存在している場合には、CPUにそれを渡す。一方、存在しない場合には、バス4を介してメインメモリ7上のデータを要求する。これにより、他のプロセッサモジュールのキャッシュメモリ、またはメインメモリ7からデータを受け取ることができるとともに、読み出しを行ったCPUのキャッシュメモリにこのデータを保持することができる。

【0022】バスコントローラ5は、プロセッサモジュール1a～1cとメインメモリ7間のデータ転送を司るものであり、メインメモリ7の更新履歴を保持するログメモリ6a、6bを制御する機能も有している。このため、バスコントローラ5は、ログメモリ6a、6bにそれぞれ対応したログカウンタ52a、52bを持つ。また、バスコントローラ5は、各プロセッサモジュール1a～1cがどのログメモリを使うべきかを決定するために使用されるログテーブル51も有している。図2には、ログテーブル51の構造が示されている。

【0023】図2に示されているように、ログテーブル51は、プロセッサモジュール1a～1cそれぞれのCPU番号とそれらCPUが現在使用しているログメモリの番号（カレントログメモリ番号）との対応関係を保持している。ここで、カレントログメモリ番号“0”はログメモリ6aを使用することを示し、カレントログメモリ番号“1”はログメモリ6bを使用することを示す。このシステムのスタートアップ時には、全てのCPUのカレントログメモリ番号は“0”を示しており、すべてのCPUはログメモリ6aを使うように設定される。

【0024】図3には、2つのログカウンタ52a、52bと2つのログメモリ6a、6bとの関係が示されている。ログカウンタ52aは、ログメモリ6aの更新履歴情報格納位置を指定するポイントを保持しており、そのポイント値は更新履歴情報がログメモリ6aに書き込まれる度に、ログメモリ6aの先頭番地から最終番地に向けて+1ずつインクリメントされる。ログカウンタ52bは、ログメモリ6bの更新履歴情報格納位置を指定するポイントを保持しており、そのポイント値は更新履歴情報がログメモリ6bに書き込まれる度に、ログメモ

り6bの先頭番地から最終番地に向けて+1ずつインクリメントされる。

【0025】通常のプログラム実行処理の期間においては、各CPUは、メインメモリ7への書き込みが必要となった時に、バスコントローラ5に対し、

- (1) CPUのID
- (2) メモリのアドレス
- (3) メモリデータ

を渡す。バスコントローラ5は、この書き込み要求を検出すると、CPUのID (CPU番号) から、どのログメモリを使うか決定し (ここではログメモリ6aを使用するものとする)、そのログメモリ6aに対応したログカウンタ52aの値を得て、そのカウンタに対応したログメモリ6aの位置に、メモリアドレスとそのメモリアドレスの更新前データとを更新履歴情報として記録する。そして、メインメモリ7を更新する。

【0026】その後、プロセッサモジュール1a~1cの各CPUは、所定の時間が経過したこと、あるいはログメモリ6aの残容量が所定の量を下回ったことをバスコントローラ5からの割り込み信号やポーリング処理によって検知したとき、チェックポイント取得処理を開始する。チェックポイント取得処理では、各CPUの内部状態復元のために必要なレジスタの値、およびキャッシュメモリ内のデータのうち、まだメインメモリ7に反映されていないデータが、バスコントローラ5を介して、メインメモリ7に書き込まれる。この場合の更新履歴についても、前述と同様にログメモリ6aに記録される。

【0027】ここまで終了したプロセッサモジュール1a~1cの各CPUは、使用するログメモリを切り替えるためにログテーブル51のカレントログメモリ番号を“0”から“1”に書き換え (ここではログメモリ6bを次に使用するものとする)、その後、即座に通常のプログラム処理を再開する。

【0028】この様にチェックポイント取得処理を完了したCPUの順で、使用するログメモリをそれまで使用していたログメモリから他方のログメモリに切り替えることにより、それまで使用していたログメモリの内容を破壊することなく、通常のプログラム実行処理を再開することができる。よって、各CPUが自身のチェックポイント取得処理を完了した時点で即座に通常のプログラム実行処理を再開できるようになり、システム全体の待機時間を大幅に減少させることができる。

【0029】次に、図4を参照して、チェックポイント取得処理全体の流れについて説明する。

【0030】いま、図4に示すように、プロセッサモジュール1a~1cのCPU2a~2cが並列に稼働しているものと、すなわち、各CPU2a~2cが、通常のプログラム実行処理をそれぞれ行なっているものとする (図4の(1))。そして、このときに使用されているログメモリはログメモリ6aであるとする (図4の

(2))。

【0031】その後、各CPU2a~2cの稼働中にログメモリ6aの残容量が予め設定された所定量を下回ったときに (図4の(3))、CPU2a~2cそれぞれは、その旨を検知して、チェックポイント処理を開始する (図4の(4))。

【0032】このチェックポイント取得処理の開始および所要時間は、上述したように、その検知タイミングや、検知したときに処理中のプログラムの種類、キャッシュメモリの状態などにより異なる。

【0033】しかし、各CPU2a~2cそれぞれは自身のチェックポイント処理が終了した際に、使用するログメモリを切り替えて、即座に通常処理を再開する (図4の(5))。これにより、通常処理を再開したCPUについては、ログメモリ6bへの更新履歴の採取が開始される (図4の(6))。

【0034】一方、すべてのCPU2a~2cのチェックポイント処理が終了した際に (図4の(7))、ログメモリ6aのリセット、すなわち、ログメモリ6aに記録された更新履歴の破棄が行なわれる。

【0035】次に、図5のフローチャートを参照して、各プロセッサモジュールのCPUが実行するチェックポイント取得処理の手順について説明する。

【0036】各プロセッサモジュール1a~1cのCPUは、通常処理を行なっているときに (ステップS

1)、記録中のログメモリの残容量が予め設定された所定量を下回ったことを検知した場合 (ステップS2)、チェックポイント取得処理を開始する (ステップS

3)。このチェックポイント取得処理では、CPUの各種レジスタを含むそのCPUの内部状態 (コンテキスト) と、メインメモリ7に反映されていないキャッシュメモリの内容がメインメモリ7に書き込まれる。次に、そのCPUは、使用するログメモリを切り替えるためにログテーブル51の該当するカレントログメモリ番号を“0”から“1”に、書き換える (ステップS4)。

その後、そのCPUは、自身がチェックポイント取得処理を完了した最後のCPUであるか否かを判断する (ステップS5)。これは、ログテーブル51を参照して、カレントログメモリ番号を書き換えてないCPUが存在するかどうかを調べることによって行われる。全てのCPUのカレントログメモリ番号が新たな番号に書き換えられていれば、そのCPUがチェックポイント取得処理を完了した最後のCPUである。この場合、そのCPUは、今まで使用していたログメモリの内容を廃棄するために、そのログメモリをクリアする (ステップS6)。

このクリア処理では、そのログメモリに対応するログカウンタのポインタ値が“0”に戻される。その後、そのCPUは、チェックポイント取得処理で中断した通常のプログラム実行処理を再開する (ステップS7)。

【0037】図1のマルチプロセッサシステムになんら

かの障害が発生した場合には、最後に使い始めたログメモリにスタック形式で保持されている更新履歴情報が逐次読み出され、更新前データがメインメモリ上の対応する番地に書き込まれる。これにより、メインメモリが障害発生前のチェックポイントの時点に復元される。もし、もう一方のログメモリについても、それに対応するログカウンタのポイント値が“0”でなければ、同様の処理が行われる。

【0038】これにより、障害以前のチェックポイントを取った時点のメインメモリの内容が復元され、そのメモリにセーブされていたレジスタなどの内容を使ってCPUの内部状態が復元できる。そして、この状態で障害の原因を解明し、それを除去してから実行を再開すれば、障害があった場合でも、システム全体に異常を起こさずに、処理を進めることが可能となる。

【0039】（第2実施形態）次に、この発明の第2実施形態に係るマルチプロセッサシステムについて説明する。このマルチプロセッサシステムは、第1実施形態に比べ、各プロセッサのキャッシュメモリのライトポリシー（ライトスルー／ライトキャッシュ）を動的に切り替える機能を有する点だけが異なり、他の点は全て第1実施形態と同じである。

【0040】ここで、キャッシュメモリと同時にメインメモリにもデータを書き込む方式をライトスルー（write through）という。この方式では、書き込みの時間はメインメモリ7のアクセス時間と同じなので、通常処理の高速化はあまり期待できない。しかし、チェックポイント取得処理でキャッシュの内容をフラッシュする必要がないので、チェックポイント取得処理に要する時間を短くすることができる。

【0041】これに対してライトバック（write back）方式では、書き込みはキャッシュメモリにだけおこなわれ、キャッシュフラッシュするときに、キャッシュメモリの内容がメインメモリに書き戻される。したがって、ライトバック方式は、ライトスルー方式の場合に比べて、チェックポイント取得処理に要する時間は長くなるものの、通常処理においては、バストラフィックを低減でき、システム全体のパフォーマンスの向上を実現できる。

【0042】したがって、このマルチプロセッサシステムでは、基本的には、全てのプロセッサモジュールのキャッシュメモリをライトバック方式で動作させておき、一定の時間内で処理を終了させることが必要なりアルタイム処理を実行する場合に、一部のプロセッサモジュールについてのみそのキャッシュメモリをライトスルー方式に切り替えて、そのプロセッサモジュールにリアルタイム処理を実行させるといった制御が行われる。このようなキャッシュのライトポリシーの切り替え処理は、このマルチプロセッサシステム上で動作するプログラムの制御下で実行される。

【0043】また、システムの稼働状態をモニタするプログラムを用いることにより、マルチプロセッサシステムのスループットおよびレスポンスタイムに基づいてライトスルー方式で動作するプロセッサモジュールの数を動的に変更することが好ましい。そのためのソフトウェア構成を図6に示す。

【0044】図6は、図1のマルチプロセッサシステムを用いて構築したトランザクション処理システムを示す。このトランザクション処理システムは、複数のクライアントコンピュータ501と、図1のマルチプロセッサシステムから構成されるサーバコンピュータ601とを有している。サーバコンピュータ601においては、複数のクライアントコンピュータ501それぞれからのトランザクション処理要求が通信プロセッサ300によって受け付けられる。それらトランザクション処理要求は、トランザクション管理プログラム200によって各種アプリケーションプログラムに振り分けられて処理される。そして、その処理結果がトランザクション管理プログラム200および通信プロセッサ300を介してクライアントコンピュータ501それぞれに通知される。

【0045】トランザクション管理プログラム200は、このトランザクション処理システムの稼働状態をモニタしており、スループット（単位時間当たりのトランザクション処理要求数）とレスポンスタイム（トランザクション処理要求が受け付けられてからそれに対する処理結果が通知されるまでの応答時間）とに基づいて、ライトスルー方式で動作するプロセッサモジュールの数を動的に変更する機能を持つ。

【0046】ここでは、図6に示されているように、図1のキャッシュメモリ3a～3cがそれぞれCPU2a～2cの一次キャッシュとして実現されている場合を例にとり、キャッシュのライトポリシー変更動作を原理を説明する。

【0047】キャッシュメモリ3a～3cそれぞれのライトポリシー（ライトバック／ライトスルー）は、対応するCPU2a～2c内部のライトポリシー制御レジスタR1～R3に設定される値によって決定される。このため、トランザクション管理プログラム200は、それらライトポリシー制御レジスタR1～R3に書き込む値を変更することによって、ライトスルー方式で動作するキャッシュを持つCPU数を制御する。この場合、どのCPUが現在ライトスルーキャッシュを持つCPUとして動作しているかについては、トランザクション管理プログラム200によって管理されている。

【0048】次に、図7を参照して、ライトスルーキャッシュとライトバックキャッシュが混在する場合のチェックポイント取得処理全体の流れについて説明する。ここでは、図4と異なる部分についてのみ説明する。

【0049】いま、CPU2a、2bの備えるキャッシュメモリをライトバックで動作させ、CPU2cの備え

るキャッシュメモリをライトスルーで動作させたとすると、CPU 2cのチェックポイント処理（図7の

（4）'）に費やす時間は、CPU 2a～2bのチェックポイント処理（たとえば図7の（4））と比較して、大幅に短縮されることになり、常に所定の時間内に終了させる必要がある処理（リアルタイム処理）などをこのCPU 2cで処理させるようにすれば、チェックポイント取得に伴う弊害を回避することが可能となる。これは、ライトスルーキャッシュを持つCPUがチェックポイント取得処理で行なわなければならないことは、そのCPUの内部状態を保存するだけとなり、キャッシュメモリの内容を書き出す処理が不要であるためである。

【0050】このため、チェックポイントの動作が長い時間継続することにより、一定時間内に動作しなければならないようなリアルタイムプロセスに悪影響を与えるといったことがほとんどなくなり、レスポンスタイムを向上することができる。

【0051】次に、図8のフローチャートを参照して、トランザクション管理プログラム200によって実行される、システムの稼働状態に応じてライトスルーキャッシュを持つCPUとして動作するCPU数を動的に変更する処理の手順について説明する。

【0052】トランザクション管理プログラム200は、まず、ライトスルーキャッシュを持つCPUとして動作するCPU（プロセッサ）数を示す変数（n）に値“0”を代入して、全てのCPU 2a～2cをライトバックキャッシュを持つCPUとして動作させる（ステップS11）。この状態でトランザクション要求の受付および応答が開始される。トランザクション管理プログラム200は、ある一定時間間隔、例えば5分単位でスループットTAおよび平均レスポンスタイムRAを求める（ステップS12）。そして、変数（n）の値を+1し、ある1つのCPUのキャッシュメモリのライトポリシーをライトバックからライトスルーに変更する（ステップS13）。そして、その後の5分間についてのスループットTBおよび平均レスポンスタイムRBを求める（ステップS14）。

【0053】次いで、トランザクション管理プログラム200は、スループットTAとスループットTBとの差分を求め、その差がある一定値 α よりも少ないか否か、つまり変数（n）を変更する前と後とでスループットがあまり変動していないかどうかを調べる（ステップS15）。スループットの変動が少ないならば、トランザクション管理プログラム200は、変数（n）を変更する前よりも変更した後の方がレスポンスタイムが減少しているかどうかを調べるために、レスポンスタイムRAからレスポンスタイムRBの値を減算し、その差分が一定値 β よりも大きいかなんかを判断する（ステップS16）。一定値 β よりも小さいならば、トランザクション管理プログラム200は、変数（n）の値を-1して、

ライトスルーキャッシュを持つCPUとして動作するCPUの数を1つ減らす（ステップS17）。一方、一定値 β よりも大きいならば、ライトスルーキャッシュを持つCPUとして動作するCPUの数は変更しない。

【0054】ステップS12～ステップS17の処理は、システムが稼働している間、繰り返し実行される。これにより、ライトスルー方式のキャッシュメモリとライトバック方式のキャッシュメモリとが適切な数で混在させることになる。

【0055】（第3実施形態）次に、この発明の第3実施形態に係るマルチプロセッサシステムについて説明する。このマルチプロセッサシステムは、第1実施形態に比べ、2つのログメモリ6a、6bが1個の物理メモリ上に実現されている点だけが異なり、他の点は全て第1実施形態と同じである。

【0056】図9には、2つのログメモリ6a、6bを1個の物理メモリ上に実現するための構成が示されている。

【0057】ログメモリ6は、通常のランダムアクセスメモリであり、0番地からM-1番地までアドレスづけがされており、ひとつの番地にひとつの更新履歴が格納できるように構成されている。このログメモリ6の記憶空間上に2つのログメモリ6a、6bが論理的に実現される。

【0058】バスコントローラ5には、前述したログテーブル51、2つのログカウンタ52a、52bに加え、メモリ制御ロジック201、マルチプレクサ202、加算器203、減算器204が設けられている。

【0059】ログカウンタ52a、52bは、それぞれ更新履歴情報を書き込むべきログメモリ6上のアドレスを保持するのに使われる。ログカウンタ52aに保持されるポインタ値はログメモリ6aを実現するために使用され、ログメモリ6aに更新履歴情報が書き込まれる度に、ログメモリ6の0番地からM-1番地に向かって加算器203によって+1ずつ順次インクリメントされる。ログカウンタ52bに保持されるポインタ値はログメモリ6bを実現するために使用され、ログメモリ6bに更新履歴情報が書き込まれる度に、ログメモリ6のM-1番地から0番地に向かって減算器204によって-1ずつ順次デクリメントされる。

【0060】したがって、0番地からログカウンタ52aが指すアドレス-1までがログメモリ6aに相当し、M-1番地からログカウンタ52bが指すアドレス+1までがログメモリ6bに相当する。メモリ制御ロジック201は、更新履歴情報（メインメモリのアドレス、更新前データ）をライトデータとしてログメモリ6に供給すると共に、書き込み要求を発行したCPU番号（プロセッサID）をマルチプレクサ202に送る。

【0061】マルチプレクサ202は、受け取ったCPU番号に対応するカレントログメモリ番号をログテーブ

17

ル51から得、そのカレントログメモリ番号に対応するログカウンタのポインタ値を選択してそれをライトアドレスとしてログメモリ6に送る。

【0062】以下、図10を参照して、図9のバスコントローラ5の動作を説明する。

【0063】初期状態では、ログカウンタ52aのポインタ値は0、ログカウンタ52bのポインタ値はM-1となっている。これは、ログメモリ6aとログメモリ6bのどちらにも更新履歴情報が格納されていないことを意味する。

【0064】CPU2a~2cのいずれかからメインメモリ7に対する書き込み要求（更新トランザクション）をバスコントローラ5が受け取ると（ステップS21）、メモリ制御ロジック20.1は、メインメモリ7から更新前データを読み込み、その更新前データとメインメモリ7のメモリアドレスとを含む更新履歴情報をログメモリ6に送る（ステップS22）。この後、マルチプレクサ20.2は、書き込み要求を発行したCPU番号に対応するカレントログメモリ番号をログテーブル51から読み取り（ステップS23）、ログカウンタ52a、52bの中で、読み取ったカレントログメモリ番号に対応するログカウンタのポインタ値を選択する（ステップS24）。これにより、その選択されたログカウンタのポインタ値が指定するログメモリ6の番地に更新履歴情報が書き込まれる。この後、その選択されたログカウンタのポインタ値が更新される（ステップS25）。

【0065】例えば、書き込み要求を発行したCPU番号に対応するカレントログメモリ番号がログメモリ6aを指定している場合には、ログカウンタ52aのポインタ値が選択され、そのポインタ値が示す番地に更新履歴情報の書き込みが行われる。そして、ログカウンタ52aから読み出されたポインタ値は、加算器20.3で+1されて、再びログカウンタ52aに格納される。同様に、書き込み要求を発行したCPU番号に対応するカレントログメモリ番号がログメモリ6bを指定している場合には、ログカウンタ52bのポインタ値が選択され、そのポインタ値が示す番地に更新履歴情報の書き込みが行われる。そして、ログカウンタ52bから読み出されたポインタ値は、減算器20.4で-1されて、再びログカウンタ52bに格納される。

【0066】次に、図11のフローチャートを参照して、この第3実施形態において各プロセッサモジュールのCPUが実行するチェックポイント取得処理の手順について説明する。

【0067】各プロセッサモジュール1a~1cのCPUは、通常処理を行なっているときに（ステップS31）、記録中のログメモリの残容量が予め設定された所定量を下回ったことを検知した場合（ステップS32）、チェックポイント取得処理を開始する（ステップS33）。ログメモリの残容量は、ログカウンタ52a

18

に保持されているポインタ値とログカウンタ52bに保持されているポインタ値の差として求めることができる。チェックポイント取得処理では、CPUの各種レジスタを含むそのCPUの内部状態（コンテキスト）と、メインメモリ7に反映されてないキャッシュメモリの内容がメインメモリ7に書き込まれる。ライトスルーキャッシュの場合には、その内容がすでにメインメモリ7に反映済みであるので、CPUの内部状態だけがメインメモリ7に書き込まれる。

【0068】次に、そのCPUは、使用するログメモリを切り替えるためにログテーブル51の該当するカレントログメモリ番号を“0”から“1”に、書き換える（ステップS34）。この後、そのCPUは、自身がチェックポイント取得処理を完了した最後のCPUであるか否かを判断する（ステップS35）。これは、ログテーブル51を参照して、カレントログメモリ番号を書き換えてないCPUが存在するかどうかを調べることによって行われる。全てのCPUのカレントログメモリ番号が新たな番号に書き換えられていれば、そのCPUがチェックポイント取得処理を完了した最後のCPUである。この場合、そのCPUは、今まで使用していたログメモリの内容を廃棄するために、そのログメモリをクリアする（ステップS36）。このクリア処理では、そのログメモリがログメモリ6aであればログカウンタ52aのポインタ値が“0”に戻され、またログメモリがログメモリ6bであればログカウンタ52bのポインタ値が“M-1”に戻される。この後、そのCPUは、チェックポイント取得処理で中断した通常のプログラム実行処理を再開する（ステップS37）。

【0069】マルチプロセッサシステムになんらかの障害が発生した場合には、最後に使い始めたログメモリにスタック形式で保持されている更新履歴情報が逐次読み出され、更新前データがメインメモリ7上の対応する番地に書き込まれる。これにより、メインメモリ7が障害発生前のチェックポイントの時点に復元される。もし、もう一方のログメモリについても、それに対応するログカウンタのポインタ値が初期値（ログカウンタ52aの場合は“0”、ログカウンタ52bの場合は“M-1”）でなければ、同様の処理が行われる。

【0070】図12には、この第3実施形態のシステムの各プロセッサモジュールが通常動作とチェックポイント処理を繰り返し行なった場合におけるログカウンタ52a、52bのポインタ値の変化の様子が示されている。

【0071】T0は初期状態であり、ログカウンタ52aのポインタ値は0、ログカウンタ52bのポインタ値はM-1である。この時、ログメモリ6a、6bには履歴情報は格納されていない。また、全てのCPUのカレントログメモリ番号はログメモリ6aを示す。この状態から通常のプログラム実行処理が開始される。

【0072】T1の時点では、ログメモリ6aの残容量、すなわちログカウンタ52bに保持されたポインタ値とログカウンタ52aに保持されたポインタ値の差が、予め設定された所定量を下回る。それを検出した各CPUは、順次チェックポイント取得処理を開始する。

【0073】T2の時点では、最初に自身のチェックポイント取得処理を終了したCPUが、ログメモリ6aからログメモリ6bへの切り替えを行ない、即座に通常のプログラム実行処理に戻る。以降、このCPUの通常処理においては、ログメモリ6bが更新履歴情報の格納に利用される。

【0074】T3の時点では、最後に自身のチェックポイント処理を終了したCPUが、ログメモリ6aからログメモリ6bへの切り替えと、ログメモリ6aの初期化を行ない、通常のプログラム実行処理に戻る。ここで、ログメモリ6aの初期化、すなわちログメモリ6aの更新履歴情報の破棄は、ログカウンタ52aのポインタ値を0に戻すことによって行われる。

【0075】T4の時点では、ログメモリ6bの残容量、すなわちログカウンタ52bに保持されたポインタ値とログカウンタ52aに保持されたポインタ値の差が、予め設定された所定量を下回る。それを検出した各CPUが、順次チェックポイント取得処理を開始する。

【0076】T5の時点では、最初に自身のチェックポイント処理を終了したCPUが、ログメモリ6bからログメモリ6aへの切り替えを行ない、即座に通常のプログラム実行処理に戻る。この通常処理においては、ログメモリ6aが更新履歴情報の格納に利用される。

【0077】T6の時点では、最後に自身のチェックポイント処理を終了したCPUが、ログメモリ6bからログメモリ6aへの切り替えと、ログメモリ6bの初期化を行ない、通常のプログラム実行処理に戻る。ここで、ログメモリ6bの初期化、すなわちログメモリ6bの更新履歴情報の破棄は、ログカウンタ52bのポインタ値をM-1に戻すことによって行われる。

【0078】T0~T2では、すべてのCPUが更新履歴をログメモリ6aに格納するため、ログカウンタ52aのポインタ値が増加し、ログカウンタ52bのポインタ値はM-1で一定である。

【0079】T2~T3では、一部のCPUが更新履歴をログメモリ6aに格納し、残りのCPUが更新履歴をログメモリ6bに格納するため、ログカウンタ52aのポインタ値が増加し、ログカウンタ52bのポインタ値は減少する。

【0080】T3の時点で、ログメモリ6aの初期化により、ログカウンタ52aのポインタ値は0となる。

【0081】T3~T5では、すべてのCPUが更新履歴をログメモリ6bに格納するため、ログカウンタ52bのポインタ値が減少し、ログカウンタ52aのポインタ値は0で一定である。

【0082】T5~T6では、一部のCPUが更新履歴をログメモリ6aに格納し、残りのCPUが更新履歴をログメモリ6bに格納するため、ログカウンタ52aのポインタ値が増加し、ログカウンタ52bのポインタ値は減少する。

【0083】T6の時点で、ログメモリ6bの初期化により、ログカウンタ52bのポインタ値はM-1となる。

【0084】以上のように、2つのログカウンタ52a, 52bのポインタ値の一方をログメモリ6の先頭番地から最終番地に向かって順次インクリメントし、他方を最終番地から先頭番地に向かって順次デクリメントすることにより、物理的に1つのログメモリ6を使って2つのログメモリ6a, 6bを実現できる。これは、単純にログメモリを2個設けた場合と比較して、ハードウェア量をほぼ半減させることができる。

【0085】(第4実施形態)次に、この発明の第4実施形態に係るマルチプロセッサシステムについて説明する。このマルチプロセッサシステムは、第1実施形態に比べ、2つのログメモリ6a, 6bが1個の物理メモリ上に実現されている点だけが異なり、他の点は全て第1実施形態と同じである。

【0086】図13には、2つのログメモリ6a, 6bを1個の物理メモリ上に実現するための構成が示されている。

【0087】ログメモリ6は、通常のランダムアクセスメモリであり、0番地からM-1番地までアドレスづけがされており、ひとつの番地で指定されるエンTRIESに更新履歴情報とログメモリ番号とを対応づけて格納できるように構成されている。すなわち、各ENTRIESは、メモリアドレス格納フィールド、更新前データ格納フィールド、ログメモリ番号格納フィールドを含む。

【0088】バスコントローラ5には、前述したログテーブル51に加え、メモリ制御ロジック301、アドレス制御ロジック302が設けられている。また、ログカウンタ52a, 52bの代わりに、3つのログカウンタ303~305と、加算器306が設けられている。

【0089】ログカウンタ303は、次に更新履歴情報を書き込むべき番地を示すポインタ値を保持するのに使われる。このログカウンタ303のポインタ値は、書き込み要求を発行したCPUがログメモリ6a, 6bのどちらをカレントログメモリとして指定している場合でも、更新履歴情報を書き込む度に、加算器306によって番地0から順に+1ずつインクリメントされる。そして、そのポインタ値がログメモリ6の最後の番地すなわちM-1に達すると、次は0に戻ってそこから再び+1ずつインクリメントされる。したがって、ログカウンタ303のポインタ値は、0, 1, ..., M-1, 0, 1, ..., M-1, 0, ...を繰り返すことになる。

【0090】ログカウンタ304は、現在使用中のログ

メモリに対応する履歴情報の中で最初のものが格納されている番地をログカウンタ305に素早く設定するためにその番地を一時的に保持する。ログカウンタ305は、現在使用中のログメモリに対応する履歴情報のうちで最初のものが格納されている番地を示すポインタを保持するのに使われる。例えば、すべてのCPUがあるチェックポイント処理を終了し、それまで使われてきたログメモリ6aに格納されている更新履歴情報が不要となった際には、次のログメモリ6bのログメモリ番号が付されている更新履歴情報の最初のものを指すように、ログカウンタ304に保持されているポインタ値がログカウンタ305に転送され、ログカウンタ305に保持されているポインタ値の更新が行なわれる。これらログカウンタ304、305は、それぞれレジスタによって構成することができる。

【0091】メモリ制御ロジック301は、書き込み要求を発行したCPU番号に対応するログメモリ番号をログテーブル51から読み取り、その読み取ったログメモリ番号と更新履歴情報とをライトデータとしてログメモリ6に供給する。アドレス制御ロジック302は、ログカウンタ303～305の制御、およびログメモリ6に与えるアドレスを制御するものであり、メモリ制御ロジック301から更新履歴情報とログメモリ番号を受け取ると、ログカウンタ303に格納されているポインタ値を読み出し、そのポインタ値をログメモリ6にライトアドレスとして与える。また、アドレス制御ロジック302は、チェックポイント取得処理を最初に完了したCPUから最初の書き込み要求が発行された時、つまりメモリ制御ロジック301から新しいログメモリ番号を受け取った時に、その時のログカウンタ303のポインタ値をログカウンタ304に転送する。さらに、アドレス制御ロジック302は、ログテーブル51を参照して全てのCPUのチェックポイント取得処理が完了したことを検知したとき、その時のログカウンタ303のポインタ値をログカウンタ304に転送する。

【0092】なお、これらログカウンタ303～305は各CPUによってリード・ライト可能なI/Oレジスタによって実現できるので、ログカウンタ間のポインタ値の転送はCPUの制御の下で行うことも可能である。

【0093】以下、図14を参照して、図13のバスコントローラ5の動作を説明する。

【0094】初期状態では、ログカウンタ303～305のポインタ値はいずれも0である。これは、ログメモリ6aとログメモリ6bのどちらにも更新履歴情報が格納されていないことを意味する。

【0095】CPU2a～2cのいずれかからメインメモリ7に対する書き込み要求（更新トランザクション）をバスコントローラ5が受け取ると（ステップS41）、メモリ制御ロジック301は、メインメモリ7から更新前データを読み込む（ステップS42）。次い

で、メモリ制御ロジック301は、書き込みを要求したCPUに対応するログメモリ番号をログテーブル51から得え、メモリアドレスおよび更新前データを含む更新履歴情報にログメモリ番号を付加してログメモリ6に送る（ステップS43）。この後、アドレス制御ロジック302は、ログカウンタ303のポインタ値をライトアドレスとしてログメモリ62与え、これによってそのポインタ値が指定するログメモリ6の番地に更新履歴情報とログメモリ番号が書き込まれる（ステップS44）。この後、ログカウンタ303のポインタ値が加算器306によって+1インクリメントされる（ステップS45）。

【0096】次に、図15のフローチャートを参照して、この第4実施形態において各プロセッサモジュールのCPUが実行するチェックポイント取得処理の手順について説明する。

【0097】各プロセッサモジュール1a～1cのCPUは、通常処理を行なっているときに（ステップS51）、記録中のログメモリの残容量が予め設定された所定量を下回ったことを検知した場合（ステップS52）、チェックポイント取得処理を開始する（ステップS53）。ここで、ログメモリの残容量は、次のように求められる。

【0098】ログカウンタ303のポインタ値をP1、ログカウンタ305のポインタ値をP2、ログメモリ6のサイズをMとすると、 $P1 > P2$ であれば、残り容量は、 $M - (P1 - P2)$ で与えられ、 $P1 > P2$ でなければ、残り容量は、 $P2 - P1$ で与えられる。

【0099】 $P1 > P2$ の場合におけるログメモリ6の利用状態とポインタP1、P2の関係を図16（A）に示す。

【0100】ログメモリ6においてログメモリ番号0が格納されているエントリがログメモリ6aとして使用される記憶領域であり、ログメモリ番号1が格納されているエントリがログメモリ6bとして使用される記憶領域である。ここでは、現在使用中のログメモリ6bに最初に更新履歴情報が格納された番地は3番地（ポインタ値P2）であり、次に更新履歴情報を格納すべき番地は1番地（ポインタ値P1）ある。もし、この状態で障害が発生した場合には、ポインタ値P1を現在の番地から3番地まで戻しながら更新履歴情報を順次取り出し、ログメモリ6bを示すログメモリ番号が付加された更新履歴情報をメインメモリ7に書き込む処理が行われる。よって、現在有効に使用されているログメモリ6bのメモリサイズは $P1 - P2$ となるので、残り容量は $M - (P1 - P2)$ となる。

【0101】 $P2 > P1$ の場合におけるログメモリ6の利用状態とポインタP1、P2の関係を図16（B）に示す。

【0102】ここでは、現在使用中のログメモリ6bに

最初に更新履歴情報が格納された番地は7番地（ポイント値P2）であり、次に更新履歴情報を格納すべき番地は3番地（ポイント値P1）ある。すなわち、ポイント値P1は、M-1番地を越え、再び0番地から現在の3番地まで増加されたことになる。もし、この状態で障害が発生した場合には、ポイント値P1を、2番地、1番地、0番地、M-1番地、……、7番地といった順序で戻しながら更新履歴情報を順次取り出し、ログメモリ6bを示すログメモリ番号が付加された更新履歴情報をメインメモリ7に書き込む処理が行われる。よって、ログメモリ6bの残りメモリサイズは、 $P2 - P1$ となる。

【0103】図15のステップS53のチェックポイント取得処理では、CPUの各種レジスタを含むそのCPUの内部状態（コンテキスト）と、メインメモリ7に反映されていないキャッシュメモリの内容がメインメモリ7に書き込まれる。ライトスルーキャッシュの場合には、その内容がすでにメインメモリ7に反映済みであるので、CPUの内部状態だけがメインメモリ7に書き込まれる。

【0104】次に、そのCPUは、使用するログメモリを切り替えるためにログテーブル51の該当するカレントログメモリ番号を“0”から“1”に、書き換える（ステップS54）。この後、そのCPUは、自身がチェックポイント取得処理を完了した最初のCPUであるか否かを判断する（ステップS55）。これは、ログテーブル51を参照して、カレントログメモリ番号を書き換えたCPUが他に存在するかどうかを調べることによって行われる。チェックポイント取得処理を完了した最初のCPUであれば、そのCPUは、ログカウンタ303のポイント値をログカウンタ304にコピーする（ステップS56）。

【0105】一方、チェックポイント取得処理を完了した最初のCPUではない場合には、そのCPUは、自身がチェックポイント取得処理を完了した最後のCPUであるか否かを判断する（ステップS57）。これは、ログテーブル51を参照して、カレントログメモリ番号を書き換えてないCPUが存在するかどうかを調べることによって行われる。全てのCPUのカレントログメモリ番号が新たな番号に書き換えられていれば、そのCPUがチェックポイント取得処理を完了した最後のCPUである。この場合、そのCPUは、ログカウンタ304のポイント値をログカウンタ305にコピーする（ステップS58）。

【0106】図17には、この第4実施形態のシステムの各プロセッサモジュールが通常動作とチェックポイント処理を繰り返し行なった場合におけるログカウンタ303～305それぞれのポイント値の変化の様子が示されている。

【0107】T0は初期状態で、ログカウンタ303～305はいずれも0を保持しており、ログメモリ6a、

6bには履歴情報が格納されていない状態である。また、全てのCPUのカレントログメモリ番号はログメモリ6aを示す。この状態から通常のプログラム実行処理が開始される。

【0108】T1の時点では、 $P1 > P2$ であるので、ログメモリ6aの残容量（ログメモリ6の残り容量と同じ）は、 $M - (P1 - P2)$ で与えられる。この残容量の値が予め設定された所定量を下回っている場合、それを検出した各CPUが、順次チェックポイント取得処理を開始する。

【0109】T2の時点では、最初に自身のチェックポイント取得処理を終了したCPUが、ログメモリ6aからログメモリ6bへの切り替えと、ログカウンタ303からログカウンタ304へのポイント値のコピーを行ない、即座に通常のプログラム実行処理に戻る。以降、このCPUの通常処理においては、ログメモリ6bが更新履歴情報の格納に利用される。

【0110】T3の時点では、最後に自身のチェックポイント取得処理を終了したCPUが、ログメモリ6aからログメモリ6bへの切り替えと、ログカウンタ304からログカウンタ305へのポイント値のコピーを行ない、即座に通常のプログラム実行処理に戻る。ログカウンタ304からログカウンタ305へのポイント値のコピーは、ログメモリ6aの履歴情報の初期化を意味する。

【0111】T5の時点では、 $P2 > P1$ であるので、ログメモリ6bの残容量は、 $P2 - P1$ で与えられる。その残り容量が予め設定された所定量を下回っている場合、それを検出した各CPUが、順次チェックポイント取得処理を開始する。

【0112】T6の時点では、最初に自身のチェックポイント処理を終了したCPUが、ログメモリ6bからログメモリ6aへの切り替えと、ログカウンタ303からログカウンタ304へのポイント値のコピーを行ない、即座に通常のプログラム実行処理に戻る。以降、このCPUの通常処理においては、ログメモリ6aが更新履歴情報の格納に利用される。

【0113】T7の時点では、最後に自身のチェックポイント処理を終了したCPUが、ログメモリ6bからログメモリ6aへの切り替えと、ログカウンタ304からログカウンタ305へのポイント値のコピーを行ない、即座に通常のプログラム実行処理に戻る。ログカウンタ304からログカウンタ305へのポイント値のコピーは、ログメモリ6bの履歴情報の初期化を意味する。

【0114】T0～T2では、すべてのCPUが更新履歴をログメモリ6aに格納するため、ログカウンタ303に保持されているポイント値が増加し、ログカウンタ304、305に保持されているポイント値は0で一定である。格納される更新履歴には、すべてログメモリ6aを示すログメモリ番号0が付加されている。

【0115】T2の時点で、ログカウンタ303に保持されているポインタ値が、ログカウンタ304に複写される。ログカウンタ305に保持されているポインタ値は変化しない。

【0116】T2～T3では、一部のCPUが更新履歴をログメモリ6aに格納し、残りのCPUが更新履歴をログメモリ6bに格納するため、格納される更新履歴には、ログメモリ6aのログメモリ番号とログメモリ6bのログメモリ番号とが混在している。ログカウンタ304、305に保持されているポインタ値は変わらない。

【0117】T3の時点で、ログメモリ6aの初期化により、ログカウンタ304に保持されているポインタ値がログカウンタ305に複写される。これにより、ログカウンタ305のポインタ値は、ログメモリ6bのログメモリ番号を付された最初の更新履歴の格納エントリを指すようになる。

【0118】T4の時点で、ログカウンタ303に保持されているポインタ値がM-1に達し、次の履歴情報の書き込みにより、0に戻る。

【0119】T3～T6では、すべてのCPUが更新履歴をログメモリ6bに格納するため、ログカウンタ303に保持されているポインタ値が増加し、ログカウンタ304、305に保持されているポインタ値は変化しない。格納される更新履歴には、すべてログメモリ6bのログメモリ番号が付されている。

【0120】T6の時点で、ログカウンタ303に保持されているポインタ値が、ログカウンタ304に複写される。ログカウンタ305に保持されているポインタ値は変化しない。

【0121】T7の時点で、ログメモリ6bの初期化により、ログカウンタ304に保持されているポインタ値がログカウンタ305に複写される。これにより、ログカウンタ305のポインタ値は、ログメモリ6aのログメモリの番号を付された最初の更新履歴の格納エントリを指すようになる。

【0122】以上示したように、第4実施形態においても、単純にログメモリを2個設けた場合と比較して、ハードウェア量をほぼ半減させることができ、従来のひとつのログメモリとさほど変わらないハードウェアで実現することが可能である。

【0123】なお、第3および第4実施形態において、ログメモリ6は、通常のランダムアクセスメモリで実現できる。このことは、ログメモリ6は、メインメモリ7の一部を使って実現できることを意味する。

【0124】一般に、マルチプロセッサシステムにおいては、それがどのような処理を実行するサーバとして使用されるか、あるいは搭載されているCPUの個数やCPU性能によって、メインメモリ7の更新の頻度や最適なチェックポイント取得の間隔が異なる。

【0125】従って、ログメモリ6をメインメモリ7の

一部を使って実現することにより、マルチプロセッサシステムが適用されるサーバの種類、CPUの個数やCPU性能、最適に設定されたチェックポイント間隔などに応じて、ログメモリ6の容量を最適に設定することが可能になる。

【0126】

【発明の効果】以上説明したように、本発明のマルチプロセッサシステムによれば、チェックポイント取得時におけるシステム全体の待機状態を大幅に減少させることが可能となり、かつ全体のパフォーマンスを低下させずに、リアルタイム処理を効率良く処理することが可能となる。また、複数のログメモリを従来のひとつのログメモリとさほど変わらないハードウェアで実現することが可能である。

【図面の簡単な説明】

【図1】この発明の第1実施形態に係るマルチプロセッサシステムの構成を示すブロック図。

【図2】図1のマルチプロセッサシステムに設けられたログテーブルの構成例を示す図。

【図3】図1のマルチプロセッサシステムに設けられた2つのログメモリと2つのログカウンタとの対応関係を示す図。

【図4】図1のマルチプロセッサシステムで実行されるチェックポイント取得処理全体の流れを示す図。

【図5】図1のマルチプロセッサシステムに設けられた各プロセッサモジュールのCPUが実行するチェックポイント取得処理の手順を示すフローチャート。

【図6】この発明の第2実施形態に係るマルチプロセッサシステムにおけるキャッシュライトポリシー制御の原理を説明するための図。

【図7】第2実施形態に係るマルチプロセッサシステムにライトスルーキャッシュとライトバックキャッシュが混在する場合のチェックポイント取得処理全体の流れを示す図。

【図8】第2実施形態に係るマルチプロセッサシステムにおいてライトスルーキャッシュを持つCPU数を変更する処理を示すフローチャート。

【図9】この発明の第3実施形態に係るマルチプロセッサシステムに設けられたバスコントローラとログメモリの構成を示す図。

【図10】図9のバスコントローラの動作を示すフローチャート。

【図11】第3実施形態のマルチプロセッサシステムに設けられた各プロセッサモジュールのCPUが実行するチェックポイント取得処理の手順を示すフローチャート。

【図12】第3実施形態のマルチプロセッサシステムに設けられた2つのログカウンタのポインタ値の変化の様子を示す図。

【図13】この発明の第4実施形態に係るマルチプロセ

27

ッサシステムに設けられたバスコントローラとログメモリの構成を示す図。

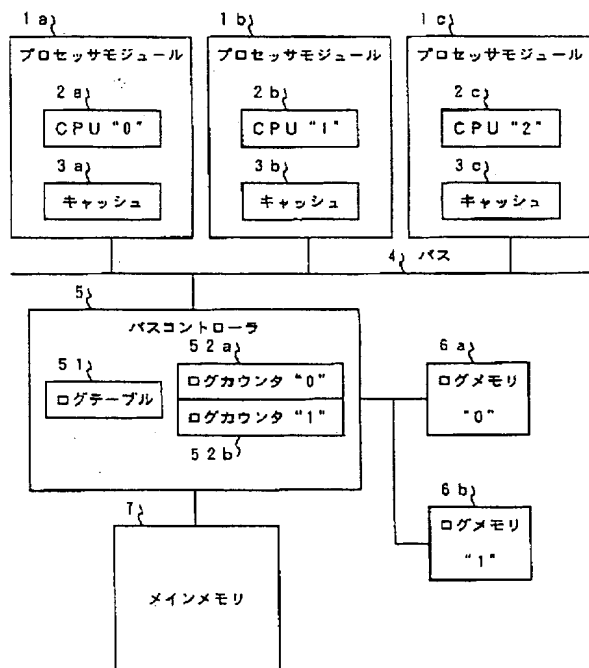
【図 1 4】図 1 3 のバスコントローラの動作を示すフローチャート。

【図 1 5】第 4 実施形態のマルチプロセッサシステムに設けられた各プロセッサモジュールの CPU が実行するチェックポイント取得処理の手順を示すフローチャート。

【図 1 6】第 4 実施形態のマルチプロセッサシステムにおけるログメモリの利用状態とその時の残り容量との関係を示す図。

【図 1 7】第 4 実施形態のマルチプロセッサシステムに

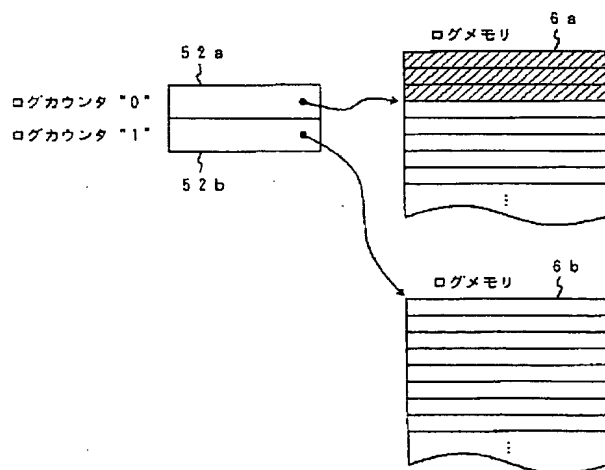
【図 1】



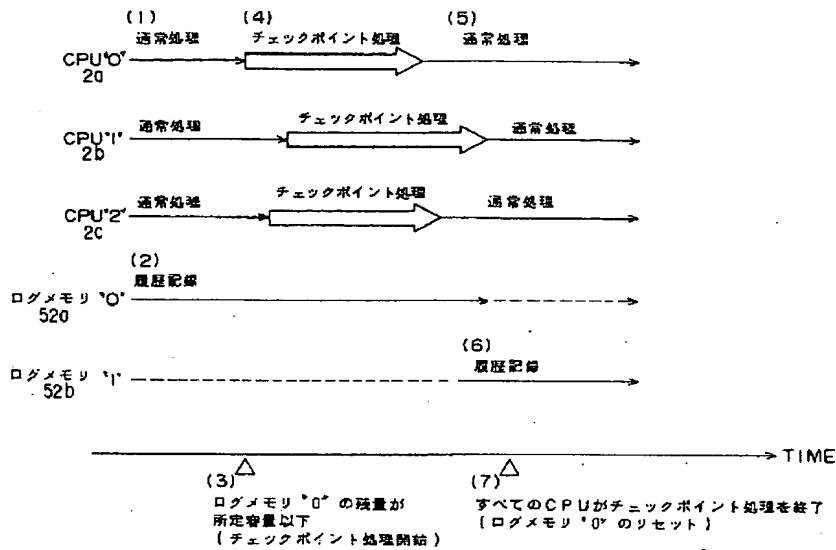
【図 2】

| ログテーブル 51 | |
|-----------|-------------|
| CPU番号 | カレントログメモリ番号 |
| 0 | 0/1 |
| 1 | 0/1 |
| 2 | 0/1 |
| ⋮ | ⋮ |

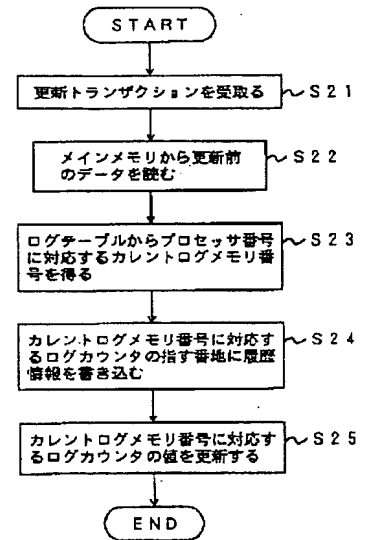
【図 3】



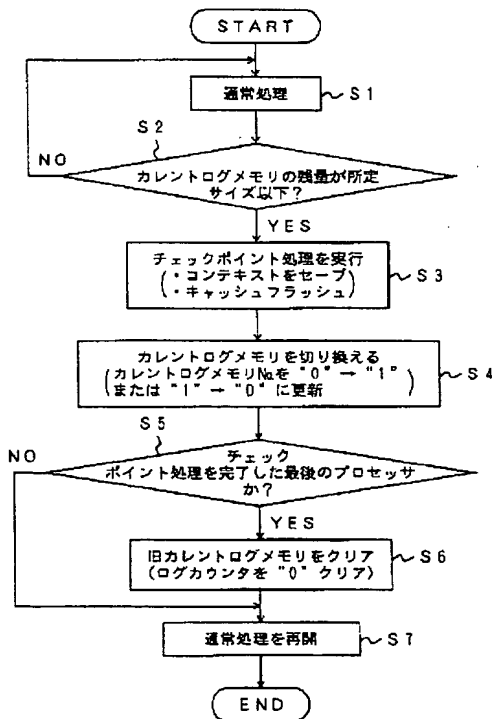
【図4】



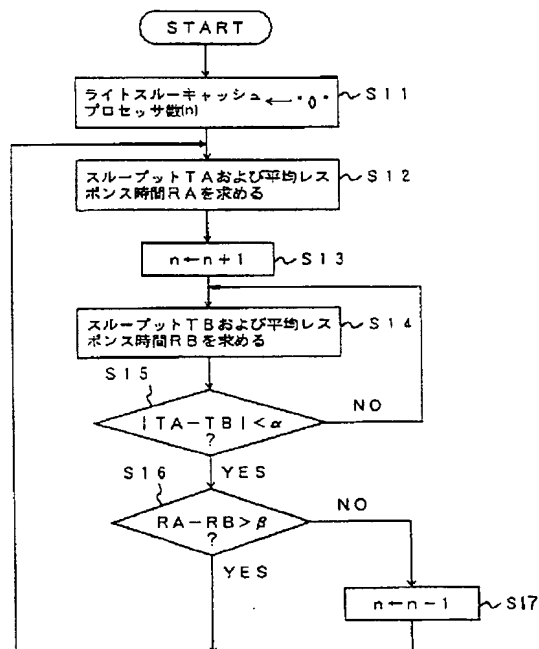
【図10】



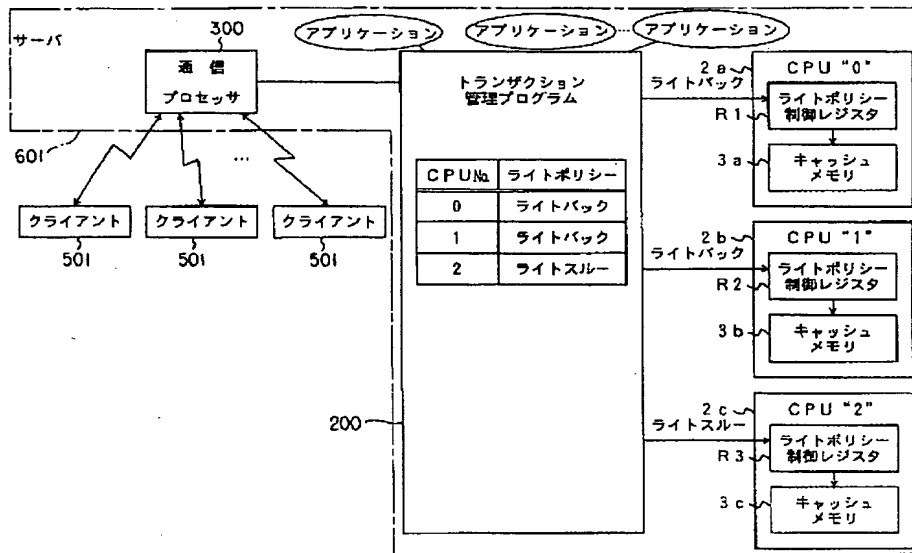
【図5】



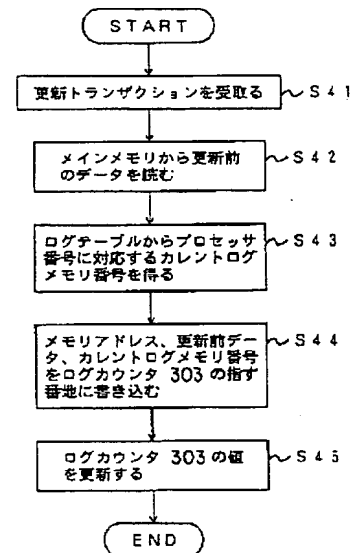
【図8】



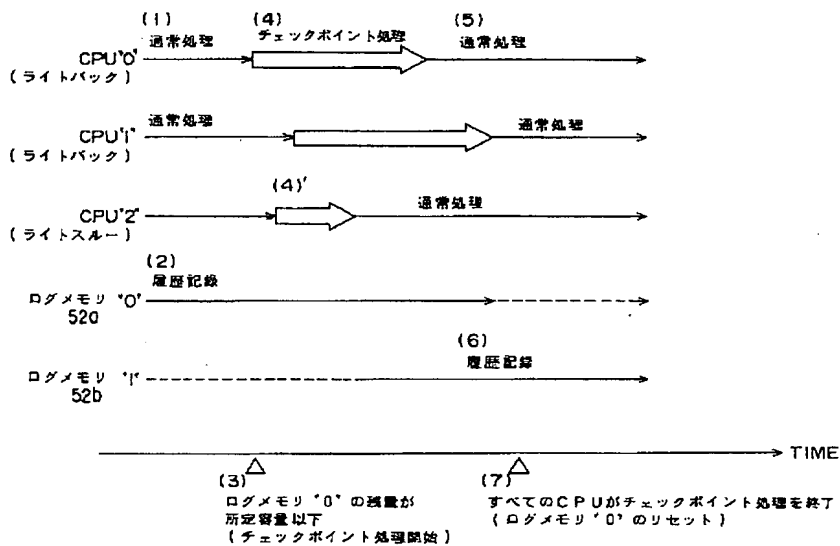
【図 6】



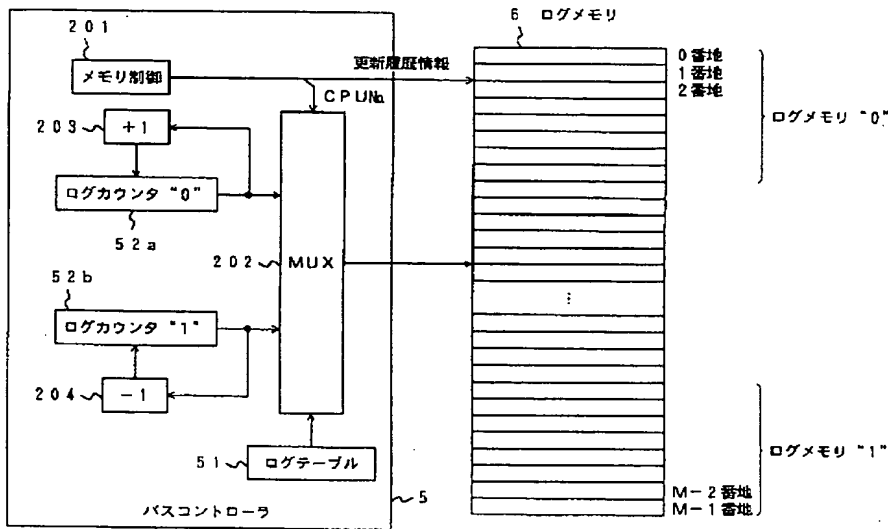
【図 14】



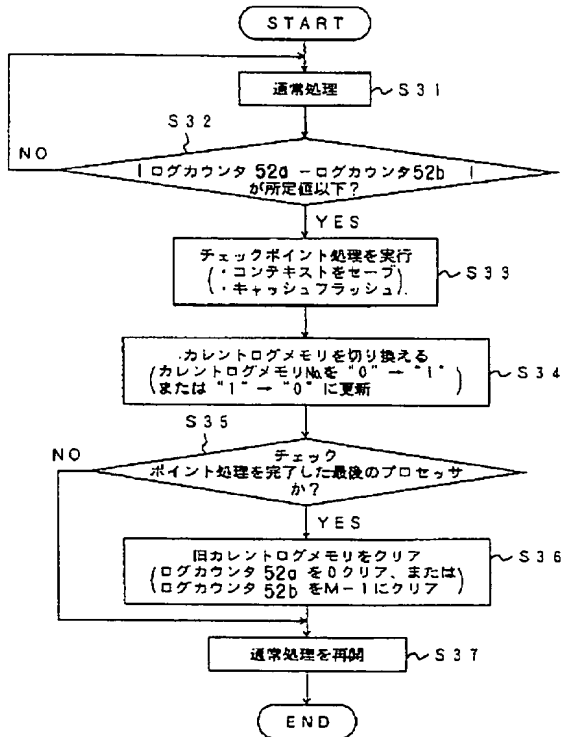
【図 7】



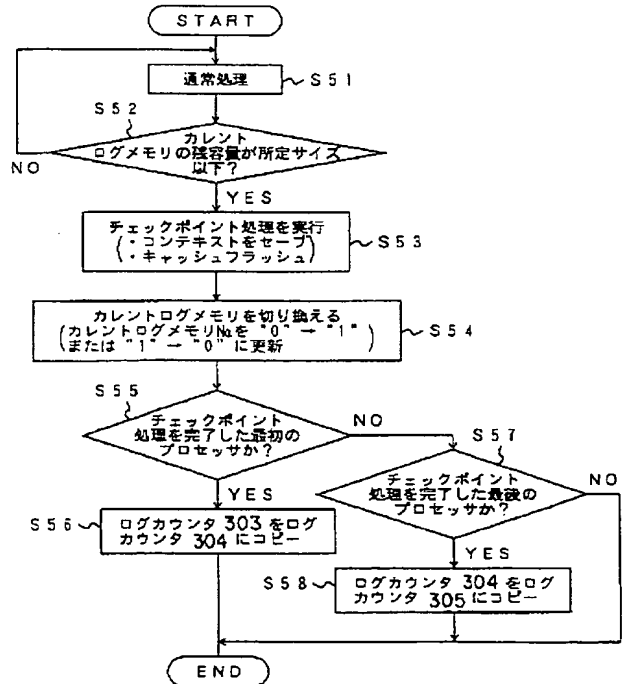
【図9】



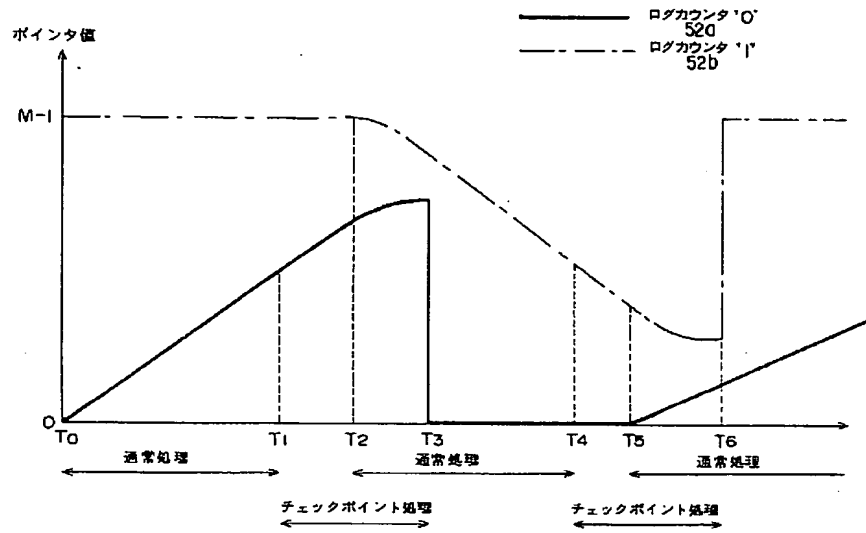
【図11】



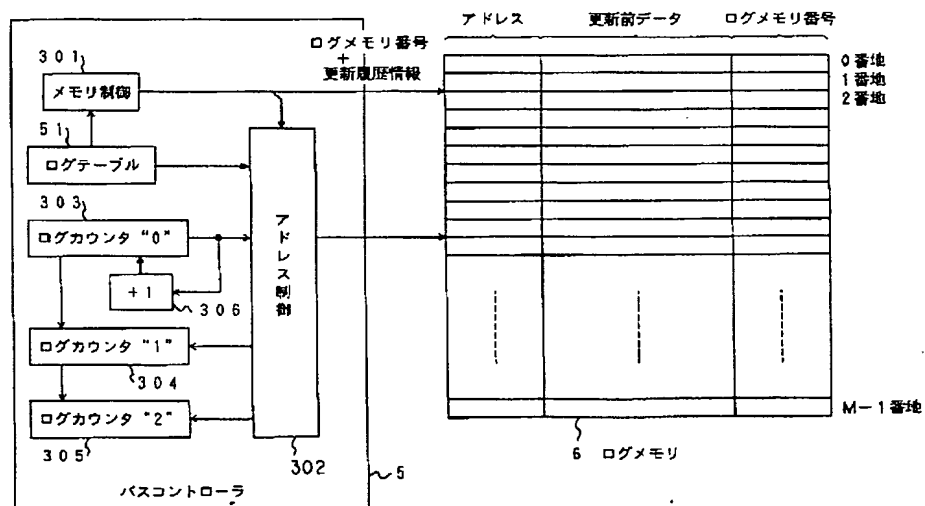
【図15】



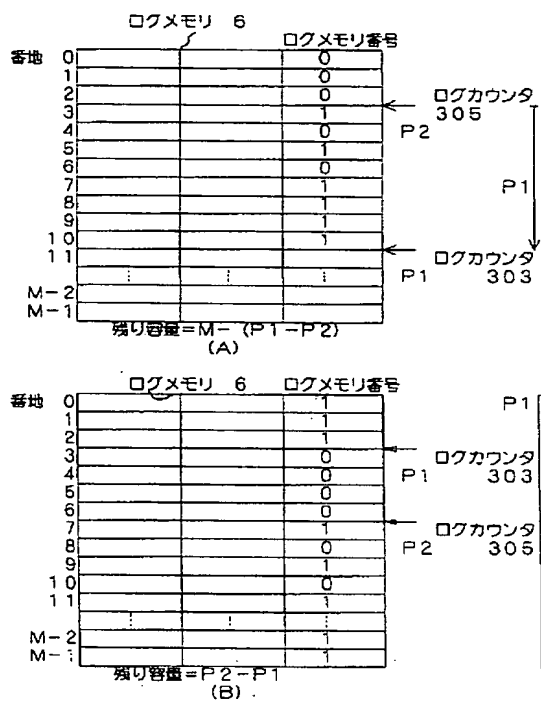
【図12】



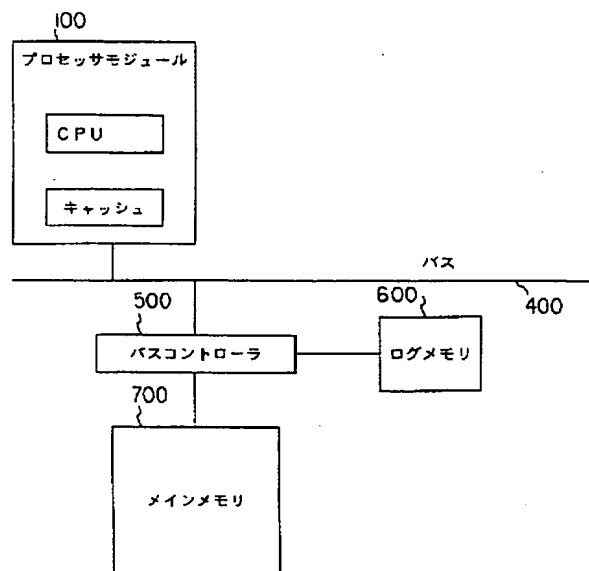
【図13】



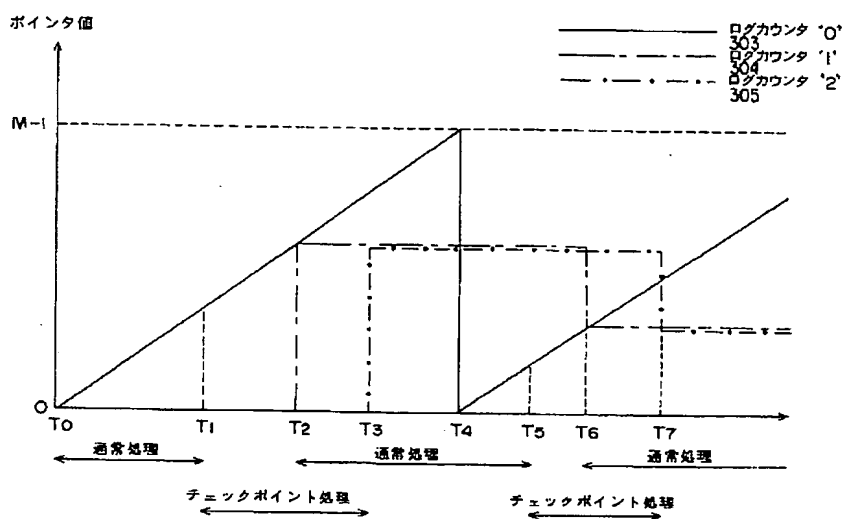
【図16】



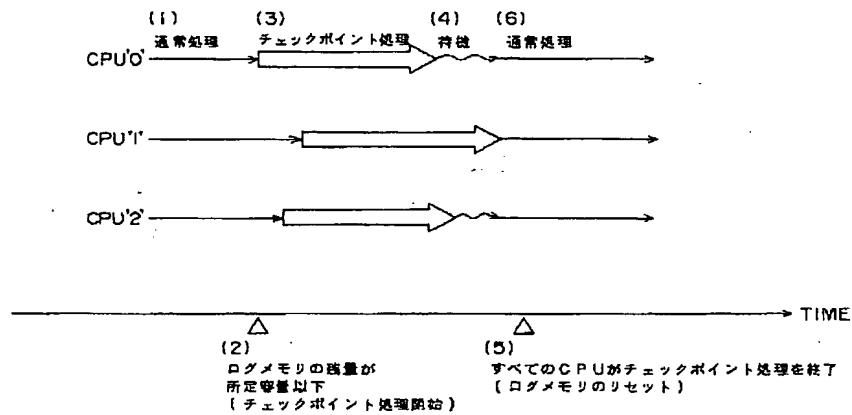
【図18】



【図17】



【図19】



フロントページの続き

(72) 発明者 大森 誉史
東京都青梅市末広町2丁目9番地 株式会
社東芝青梅工場内

(72) 発明者 増渕 美生
東京都青梅市末広町2丁目9番地 株式会
社東芝青梅工場内

(72) 発明者 藤井 高広
東京都青梅市末広町2丁目9番地 株式会
社東芝青梅工場内

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.